同じアソシエーションスキームを作る群の計算 (Computer Algebra : Design of Algorithms, Implementations and Applications)

AUTHOR(S):

宮本, 泉

数理解析研究所講究録
第1666巻 2009年 127-134

同じアソシエーションスキームを作る群の計算

宮本 泉*
IZUMI MIYAMOTO
山梨大学
UNIVERSITY OF YAMANASHI

1 Introduction

可移な置換群は、自己同型群が可移なアソシエーションスキームを作る。このようなアソシエーションスキームはschurianであると呼ばれている。schurianなアソシエーションスキームは、その自己同型群からばかりでなく、通常、複数の可移な置換群からも同じものが得られる。本講演では、schurianなアソシエーションスキームから、それを作る可移な置換群のすべてを計算することを考える。31次までの可移な置換群は分類されている[5]ので、これを使えば、どの可移な群がどのアソシエーションスキームを作るかは分る。アソシエーションスキームの分類は、32次[4]でもできているので、もし、それが計算可能な範囲内であれば、32次の可移な置換群のすべてを作ることができることになる。

以上のような考えで計算を始めたが、本発表の後、2009年1月に32次の可移置換群の分類結果[1]が発表された。その結果によると、約280万個の群が存在するということである。本実験では、その時点で約30万個の群が得られていて、群から作られる4000個ほどのアソシエーションスキームのうち残っているのは150個ほどであったので、これらのアソシエーションスキームから250万個の群が出てくるということになる。

したがって、本発表の意義は薄れてしまった。しかし、可移置換群の分類では、今までの経緯から、なんらかの再検証は必要であることは明白となっている。また、いずれの文献[5, 1]でも、分類をする具体的なコンピュータプログラムは示されていない。そこで、ここでは、数式処理ソフトウェアGAPシステム[2]に必要な関数はほとんどそろっていることを示し、本発表のもう1つの目的であったGAPの関数の使い方の紹介を中心に、具体的な説明を行うことにする。

2 GAPによる置換群の基本的な計算

```
gap> G:=Group([(1,2,7,5,10,11)(3,9,6)(4,8), (1,2,12)(3,7,11)(4,5,6)(8,9,10)]);;
gap> IsPermGroup(G); #Gは置換群？
true
gap> MovedPoints(G); #Gの動かす点集合
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ]
gap> Orbit(G,1); #点1とGの作用で移りあう点集合
[ 1, 11, 12, 10, 7, 2, 5, 9, 3, 4, 8, 6 ]
```

*imiyamoto@yamanashi.ac.jp

```
gap> Orbits(G); #G で互いに移りあう点の集合の全体
[ [ 1, 2, 7, 12, 5, 11, 10, 6, 3, 8, 4, 9 ] ]
gap> IsTransitive(G,[1..12]); #可移 (Orbits=[MovedPoints])?
true
gap> G1:=Stabilizer(G,1);; #点 1 の固定部分群
gap> Orbits(G1);
[ [ 2, 6, 12, 8, 11, 3 ], [ 4, 7, 10 ], [ 5, 9 ] ]
gap> IsTransitive(G1,[1..12]);
false
gap> Orbit(G,[1,5],OnSets); #点集合の点ごとに作用
[ [ 1, 5 ],[ 2, 10 ],[ 2, 6 ],[ 7, 11 ],[ 8, 12 ],[ 3, 7 ],[ 4, 12 ],[ 3, 11 ],
  [ 1, 9 ],[ 5, 9 ],[ 6, 10 ],[ 4, 8 ] ]
gap> Orbit(G,[1,5],OnTuples); #点対に作用
[ [ 1, 5 ],[ 2, 10 ],[ 2, 6 ],[ 7, 11 ],[ 12, 8 ],[ 7, 3 ],[ 12, 4 ],[ 5, 1 ],
  [ 11, 3 ],[ 1, 9 ],[ 5, 9 ],[ 11, 7 ],[ 10, 2 ],[ 6, 2 ],[ 3, 7 ],[ 10, 6 ],
  [ 6, 10 ],[ 3, 11 ],[ 8, 12 ],[ 4, 12 ],[ 9, 5 ],[ 8, 4 ],[ 4, 8 ],[ 9, 1 ] ]
```

3 置換群の作るアソシエーションスキーム

定義 (アソシエーションスキーム)
$\{R_k\}_{k=1,2,\cdots,d}$ が点集合 $\{1,2,\cdots,n\}$ 上のアソシエーションスキームであるとは、

AS1. $\{R_t\}_{t=1,2,\cdots,d}$ は、$\{1,2,\cdots,n\} \times \{1,2,\cdots,n\}$ の分割

AS2. $R_1 = \{(1,1),(2,2),\cdots,(n,n)\}$

AS3. $R_{t^*} = \{(j,i)|(i,j) \in R_t\}$

AS4. どの $(i,k) \in R_u$ に対しても、次の点の個数 $p_{stu} = \#\{j|(i,j) \in R_s, (j,k) \in R_t\}$ は一定

Transitive な置換群の点対 $\{[i,j]\}$ 上の orbit 全体を R_0, R_1, $\cdots$, R_d とするとアソシエーションスキームとなる。orbit ごとに番号 t を付けて、それを成分とする行列で表すと下のようになる。
[例] 下の 6 点上の置換群 G で、点対 $\{[i,j]\}$ 上の orbit が 4 個のとき、$t = 0,1,2,3$ として、

```
gap> G:=WreathProduct(Group((1,2,3)),Group((1,2)))^(1,2);
Group([ (1,3,2), (4,5,6), (1,5)(2,4)(3,6) ])
gap> Orbits(G,Tuples([1..6],2),OnTuples);
[ [ [ 1, 1 ], [ 3, 3 ], [ 5, 5 ], [ 2, 2 ], [ 6, 6 ], [ 4, 4 ] ],
  [ [ 1, 2 ], [ 3, 1 ], [ 5, 4 ], [ 2, 3 ], [ 6, 5 ], [ 4, 6 ] ],
  [ [ 1, 3 ], [ 3, 2 ], [ 5, 6 ], [ 2, 1 ], [ 6, 4 ], [ 4, 5 ] ],
  [ [ 1, 4 ], [ 3, 4 ], [ 1, 5 ], [ 5, 2 ], [ 2, 4 ], [ 3, 5 ], [ 6, 2 ],
      [ 1, 6 ], [ 5, 1 ], [ 2, 5 ], [ 4, 2 ], [ 3, 6 ], [ 6, 1 ], [ 5, 3 ],
      [ 2, 6 ], [ 4, 1 ], [ 6, 3 ], [ 4, 3 ] ] ]
```

$$A = \begin{pmatrix} 0 & 1 & 2 & 3 & 3 & 3 \\ 2 & 0 & 1 & 3 & 3 & 3 \\ 1 & 2 & 0 & 3 & 3 & 3 \\ 3 & 3 & 3 & 0 & 2 & 1 \\ 3 & 3 & 3 & 1 & 0 & 2 \\ 3 & 3 & 3 & 2 & 1 & 0 \end{pmatrix}$$

4 Transitive な群の性質

```
gap> AllBlocks(G); #MovedPoints の分割を与えるのが block
[ [ 1, 4, 7, 10 ], [ 1, 5, 9 ] ]
gap> Orbit(G,[1,4,7,10],OnSets); #その分割
[ [ 1, 4, 7, 10 ], [ 2, 5, 8, 11 ], [ 3, 6, 9, 12 ] ]
gap> IsPrimitive(G); #G は原始的 (block 無し) ?
false
```

GAP システムのライブラリ

- primitive な置換群　2499 点まで (primitive $\Longleftrightarrow$ 1 点の固定部分群が極大)
- transitive な置換群　30 点まで

アソシエーションスキームの分類 [4] は、30 点までと 32 点、33 点、34 点、38 点で、できている。

それぞれの個数

点数 (次数)	primitive な置換群	transitive な置換群	assoc. scheme	schurian scheme
18	4	983	95	93
19	8	8	7	6
20	4	1117	95	95
21	9	164	32	32
22	4	59	16	16
23	7	7	22	4
24	5	25000	750	669
25	28	211	45	32
26	7	96	34	24
27	15	2392	502	122
28	14	1854	185	124
29	8	8	26	6
30	4	5712	243	228
31	12	12	?	8
32	7	2801324*	18210	4261

Transitive な置換群の分類は、30 次までは、A.Hulpke[5] 他によりなされている。31 次は素数次なので、可移な群は primitive になる。32 次* は、J. J. Cannon と D. F. Holt[1] による。それらに書かれている分類

方法の概要は、以下の通りである。
群は、imprimitive なもののみを考えれば良い。Gに、サイズℓのblockがあると、Sym_k =SymmetricGroup(k)とおいて、

$$G \subseteq \mathsf{WreathProduct}(Sym_\ell, Sym_{n/\ell})$$

となることが知られている [3]。これにしたがって、次数の小さな transitive group から順に分類していく。
対称群から、次々と極大部分群を計算するには、GAP の関数
ConjugacyClassesMaximalSubgroups(G) 極大部分群の共役類
MaximalSubgroupClassReps(G)　その代表元
が、利用できる。アソシエーションスキームの自己同型群にこの関数のみを適用して、同じアソシエーションスキームを作る群を計算した結果、11 次まではすべて計算できた。最初に計算困難だったのは、12 次で、2 個の 6 次対称群の WreathProduct となる場合で、1 時間程度かかった。
群 G において、すべての block を固定するような部分群は、G の 1 つの block への作用の群の直積ばかりではなく、関数 SubdirectProducts で得られる群のどれかになっている。ここで、すべての block を固定するような部分群の 1 つの block への作用と G の 1 つの block への (その集合としての固定群の) 作用の違いについて考慮する必要があることを注意しておく。

```
gap> Sym:=function(k) return SymmetricGroup(k);end;
function( k ) ... end
gap> G:=WreathProduct(Sym(4),Sym(3));
<permutation group of size 82944 with 8 generators>
gap> (1*2*3*4)^3*(1*2*3);
82944
gap> O:=Orbit(G,[1..4],OnSets);
[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ], [ 9, 10, 11, 12 ] ]
gap> hom:=ActionHomomorphism(G,O,OnSets);;
gap> D:=DirectProduct(Sym(4),Sym(4),Sym(4));;
gap> Image(hom)=Sym(3);Kernel(hom)=D;
true
true
gap> SD2:=SubdirectProducts(Sym(4),Sym(4));
[ Group([ (1,2,3,4)(5,6,7,8), (1,2)(5,6) ]),
  Group([ (1,2), (1,3), (1,3,4), (5,6), (6,7), (5,7,8) ]),
  Group([ (1,2)(3,4), (1,4)(2,3), (1,2,3), (5,6)(7,8),
       (5,7)(6,8),   (5,6,7), (3,4)(7,8) ]),
  Group([ (1,2)(3,4), (1,4)(2,3), (5,6)(7,8), (5,7)(6,8),
      (3,4)(7,8),   (2,4,3)(6,8,7) ]) ]
gap> SD3:=List(SD2,u->SubdirectProducts(u,Sym(4)));;
gap> time;(ミリ秒。SD4,SD5 と作っていくと、非常に時間がかかるようになる。)
9145
gap> List(SD3,Length);
[ 4, 8, 7, 6 ]  #全部で 25 個
gap> SD3:=Concatenation(SD3);;
gap> List(SD3,u->PositionProperty(SD3,v->IsConjugate(G,u,v)));
```

```
[ 1, 2, 3, 4, 2, 6, 7, 8, 7, 10, 8, 2, 3, 7, 15, 16, 16, 18, 3, 4, 8, 16, 23, 4, 25 ]
(共役なものは、そのうちの 1 つだけを考えれば良い)
gap> Set(last);
[ 1, 2, 3, 4, 6, 7, 8, 10, 15, 16, 18, 23, 25 ]
gap> List(SD3{last},u->IsSubgroup(D,u));  ちょっと確認
[ true, true, true, true, true, true, true, true, true, true, true, true ]
```

(注) 上の共役をとる計算は、G で行うのではなく、以下に出てくる MaximalSubgroup ごとに行わなければならない。

この中から、Normalizer(正規化群) が transitive なものを選ぶ。
同じことを、Sym_4 の primitive 部分群と Sym_3 の transitive 部分群に対して行う必要がある。

5 Image(hom) の部分群と Kernel(hom) の SD3 の群との結び付け方

```
gap> MAX:=MaximalSubgroupClassReps(Image(hom));
[ Group([ (1,2,3) ]), Group([ (2,3) ]) ]
gap> List(MAX,u->IsTransitive(u,MovedPoints(Image(hom))));
[ true, false ]
gap> MAX:=Filtered(MAX,u-> IsTransitive(u,MovedPoints(Image(hom))));
[ Group([ (1,2,3) ]) ]
  (繰返し行って、Image(hom) の可移部分群のすべてを求める)
gap> MAX:=List(MAX,u->PreImage(hom,u));(G の中への引戻し)
[ <permutation group with 11 generators> ]
gap> MAX:=Concatenation([G],MAX);;
    (最初に与えた群 G も含めて、次の処理へ)
gap> NSD3:=List(MAX,v->List(SD3,u->Normalizer(v,u)));;
gap> List(NSD3,u->List(u,v->IsTransitive(v,[1..12])));
[ [ true, false, false, false,.. 中略..true, false, true ],
  [ true, false,.. 中略.. true, false, true ] ]
gap> NSD3:=List(NSD3,u->Filtered(TransposedMat([u,SD3]),v->IsTransitive(v[1],[1..12])));;
gap> NSD3[1][3];(SubdirectProduct とその正規化群の組 (逆順で))
[ <permutation group of size 82944 with 14 generators>,
  <permutation group of size 6912 with 11 generators> ]
gap> List(NSD3,u->List(u,v->IsTransitive(v[1],[1..12])));
[ [ true, true, true, true, true, true, true ],
  [ true, true, true, true, true, true, true ] ]  (確認)
gap> List([1..Length(MAX)],u->List(NSD3[u], v->Image(hom,v[1])=Image(hom,MAX[u])));
[ [ true, true, true, true, true, true, true ],
  [ true, true, true, true, true, true, true ] ] (確認 OK)
gap> List([1..Length(MAX)],u->List(NSD3[u],v->
>                  PositionProperty(NSD3[u],w->IsConjugate(MAX[u],v[2],w[2]))));
[[ 1, 2, 3, 4, 5, 6, 7 ],[ 1, 2, 3, 4, 5, 6, 7 ]] (確認 OK)
```

```
gap> Y:=NSD3[1][3];;
gap> nhom:=NaturalHomomorphismByNormalSubgroup(Y[1],Y[2]);
[ (6,8)(9,10), 中略    (1,5,10,4,7,9,3,8,12,2,6,11) ] ->
[ <identity> of ..., <identity> of ..., 中略 f2^2*f3 ]

Y[1] の hom による像と SubdirectProduct の群 Y[2] を結びつる。
gap> Intersection(Y[1],Kernel(hom))=Y[2];(注:Kernal(hom)=D)
false
しかし、Y[1] のままでは直接結びついていないので、補群を計算
(現時点では、正規部分群が solvable の場合のみ動くアルゴリズム)
gap> Complementclasses(Image(nhom),Image(nhom,Intersection(Y[1],D)));
[<pc group with 2 generators>,<pc group with 2 generators>]
gap> compl:=List(last,u->PreImage(nhom,u));(G 中への引戻し)
[ <permutation group with 13 generators>, <permutation group with 13 generators> ]

最終確認
gap> List(compl,u->IsTransitive(u,[1..12]));
[ true, true ]
gap> IsConjugate(SymmetricGroup(12),compl[1],compl[2]);
false
gap> List(compl,TransitiveIdentification);
[ 290, 291 ]
gap> List([1,2],u->RepresentativeAction(SymmetricGroup(12),
>                                  TransitiveGroup(12,last[u]),compl[u]));
[ (2,5,6,10,4)(3,9,11,8,7), (2,5,6,10,4)(3,9,11,8,7) ]
gap> List([1,2],u->TransitiveGroup(12,last2[u])^last[u]=compl[u]);
[ true, true ]
```

6 SubdirectProduct を使わない方法 (Kernel が可解な場合)

少しの例外を除いて、極小正規部分群は elementary abelian となる。本実験では、SubdirectPriduct の場合は後に廻して、下に示す elementary abelian group に作用したときの不変部分群を計算する関数を利用した。文献 [5] にはこの方法への言及は無いが、文献 [1] では、16 次の極小可移置換群の不変にする位数 2^{16} の elementary abelian 2-group の部分群をあらかじめ求めた後に、その正規化群を利用して計算を行っている。

```
gap> G1:=Stabilizer(G,O[1],OnSets);
<permutation group of size 27648 with 13 generators>
gap> K1:=List(GeneratorsOfGroup(G1), u->RestrictedPerm(u,O[1]));
[ ()...., (), (3,4), (2,4,3), (1,3)(2,4), (1,4)(2,3) ]
gap> K1:=Group(K1);
Group([ (), ...(3,4), (2,4,3), (1,3)(2,4), (1,4)(2,3) ])
gap> EAS:=ElementaryAbelianSeries(K1);
```

```
[ Group([ (3,4), (2,3,4), (1,3)(2,4), (1,4)(2,3) ]),
  Group([ (2,3,4), (1,3)(2,4), (1,4)(2,3) ]),
  Group([ (1,3)(2,4), (1,4)(2,3) ]), Group(()) ]
gap> EAS:=List(EAS,u->NormalClosure(G,u));;
gap> EAS:=List(EAS,u->Intersection(u,D));;
gap> MAX1:=MAX[1];
<permutation group of size 82944 with 8 generators>
gap> hom2:=NaturalHomomorphismByNormalSubgroup(MAX1,EAS[2]);
[ (1,2,3,4), (1,2), ... 中略..., (1,5)(2,6)(3,7)(4,8) ] ->
[ f5, f5, f4, f4, f3, f3, f2, f1*f2 ]
gap> conj:=List(GeneratorsOfGroup(MAX1),u->
>       ConjugatorIsomorphism(Image(hom2,EAS[1]),Image(hom2,u)));
[ ^f5, ^f5, ^f4, ^f4, ^f3, ^f3, ^f2, ^f1*f2 ]
gap> INVS:=InvariantSubgroupsElementaryAbelianGroup(Image(hom2,EAS[1]),conj);
[ Group([  ]), Group([ f3*f4*f5 ]), Group([ f3*f5, f4*f5 ]), Group([ f3, f4, f5 ]) ]
gap> INVS:=List(INVS,u->PreImage(hom2,u));
[ <permutation group with 9 generators>,.. 中略 ..
    ...,   <permutation group with 12 generators> ]
gap> nhoms:=List(INVS,u->
>     NaturalHomomorphismByNormalSubgroup(Image(hom2),u));
[ IdentityMapping( Group([ f1, f2, f3, f4, f5 ]) ),
  [ f1, f2, f3, f4, f5 ] -> [ f1, f2, f3*f4, f3, f4 ],
  [ f1, f2, f3, f4, f5 ] -> [ f1, f2, f3, f3, f3 ],
  [ f1, f2, f3, f4, f5 ] -> [ f1,. 中略.<identity> of...] ]
gap> compls:=List(nhoms,u->Complementclasses(Image(u),Image(u,Image(hom2,EAS[1]))));
[ [ <pc group with 2 generators>,  <pc group with 2 generators> ],   ... 中略...
    [ Group([ f1, f2, .. 中略.., <identity> of ... ]) ] ]
gap> compls:=List(nhoms,u->List(Complementclasses(
>             Image(u),Image(u,Image(hom2,EAS[1]))),w->PreImage(u,w)));
[ [ <pc group with 2 generators>, <pc group with 2 generators> ],   ... 中略...
    [ Group([ f3, f4, f5, f1, f2 ]) ] ]
gap> List(last,u->List(u,v->IsSubgroup(Image(hom2),v)));
[ [ true, true ], [ true ], [ true, true ], [ true ] ]
gap> compls:=List(nhoms,u->List(Complementclasses(
>      Image(u),Image(u,Image(hom2,EAS[1]))),w->PreImage(hom2,PreImage(u,w))));
[ [ <permutation group with 11 generators>, <permutation group with 11 generators> ],
  [ <permutation group with 12 generators> ],
  [ <permutation group with 13 generators>, <permutation group with 13 generators> ],
  [ <permutation group with 14 generators> ] ]
```

(確認作業)

```
gap> complsc:=Concatenation(compls);;
gap> List(complsc,u->PositionProperty(complsc,v->IsConjugate(G,u,v)));
```

```
[ 1, 2, 3, 4, 5, 6 ]
gap> List(complsc,TransitiveIdentification);
[ 275, 276, 283, 290, 291, 294 ]
```

7 アソシエーションスキームの利用

すべての transitive group を生成するには大ざっぱなので、出来上がった分類のすべての群の Maximal-Subgroup を計算して確認 (A.Hulpke[5]) している。本実験では、1 つのアソシエーションスキームを作る群毎に分類する。アソシエーションスキームの自己同型群から始めて、その部分群で同じアソシエーションスキームを作るものを計算して行く。アソシエーションスキームの自己同型群 G が transitive なとき、G の transitive な部分群 H で、G と H の 1 点固定部分群が一致するものが同じアソシエーションスキームを構成する。

SubdirectProduct を使う場合を、まだ、取り扱っていないが、残りの部分についての大体の計算時間は、22 次までは、かかっても数時間、24 次から 28 次は 1 日程度、30 次のとき 3 日程度であった。どの次数でも、計算に長時間を要するのは、ごく少数の場合で、ほとんど全ての場合は、短時間で計算できている。GAP ライブラリの transitive な群を参照すると、おおむねプログラムのバグは無くなって計算できていることがわかる。しかし、後まわしにして残っている場合が、どの程度の困難さであるか、予断はできない。この方法で、32 次の場合の検証をするための実験を続けていて、メモリの使用量や計算に時間のかかる部分などを調べて、改良を進めている。

参 考 文 献

[1] J. J. Cannon and D. F. Holt. The transitive permutation groups of degree 32. *Experiment. Math.* 17-3:307-314, 2008.

[2] The GAP Groups. Gap - groups, algorithms and programming, version 4. *Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany and School of Mathematical and Computational Sciences, Univ. St. Andrews, Scotland*, 1997.

[3] M. Krasner, L. [A.] Kaloujnine. Produit complet des groupes de permutations et problème d'extension de groupes II. *Acta Sci. Math. (Szeged)*, 14 39–66, 1951.

[4] A. Hanaki, I. Miyamoto. `http://kissme.shinshu-u.ac.jp/as/`

[5] A.Hulpke. Constructing Transitive Permutation Groups. *J. Symbolic Comput.*, 39 1–30, 2005.